TRANSMITTAL OF APPEAL BRIEF (Large Entity)

Docket No.
NCR.0038US

In Re Application Of: **Gang Luo et al.**

Application No.	Filing Date	Examiner	Customer No.	Group Art Unit	Confirmation No.
09/842,991	**04-26-2001**	**Chongshan Chen**	**21906**	**2172**	

Invention: **Method and Apparatus for Performing Hash Join**

COMMISSIONER FOR PATENTS:

Transmitted herewith in triplicate is the Appeal Brief in this application, with respect to the Notice of Appeal filed on

The fee for filing this Appeal Brief is: **$340.00**

☐ A check in the amount of the fee is enclosed.

☐ The Director has already been authorized to charge fees in this application to a Deposit Account.

☒ The Director is hereby authorized to charge any fees which may be required, or credit any overpayment to Deposit Account No. **50-1673** (9558)

☐ Payment by credit card. Form PTO-2038 is attached.

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

Signature

Dated: **October 27, 2004**

Dan C. Hu
Registration No. 40,025
TROP, PRUNER & HU, P.C.
8554 Katy Freeway, Suite 100
Houston, TX 77024
Telephone: (713) 468-8880, ext. 304
Facsimile: (713) 468-8883

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to "Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450" [37 CFR 1.8(a)] on

10-27-2004
(Date)

Ginger Yount
Signature of Person Mailing Correspondence

Ginger Yount
Typed or Printed Name of Person Mailing Correspondence

cc:

AF / IW

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	GANG LUO ET AL.	§	Group Art Unit:	2172
Serial No.:	09/842,991	§	Examiner:	Chongshan Chen
Filed:	April 26, 2001	§		
For:	METHOD AND APPARATUS FOR PERFORMING HASH JOIN	§	Atty. Dkt. No.:	NCR.0038US (9558)

Mail Stop **Appeal Brief-Patents**
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF PURSUANT TO 37 C.F.R § 41.37

Sir:

The final rejection of claims 1-35 is hereby appealed.

I. REAL PARTY IN INTEREST

The real party in interest is the NCR Corporation by virtue of the assignment recorded at reel/frame 11772/0068-70.

II. RELATED APPEALS AND INTERFERENCES

None.

Date of Deposit: October 27, 2004

I hereby certify under 37 CFR 1.8(a) that this correspondence is being deposited with the United States Postal Service as **first class mail** with sufficient postage on the date indicated above and is addressed to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313.

Ginger Yount

Ginger Yount

III. STATUS OF THE CLAIMS

Claims 1-35 have been finally rejected and are the subject of this appeal.

IV. STATUS OF AMENDMENTS

No amendments have been submitted after final rejection.

V. SUMMARY OF THE CLAIMED SUBJECT MATTER

Independent claim 1 recites a method that comprises storing first tuples in a first table in a database system, storing second tuples in a second table in the database system, partitioning the first and second tuples into plural portions, redistributing the first and second tuples to plural nodes according to the partitioning, and hash joining the first and second tuples to produce result tuples as the first and second tuples are being redistributed to the plural nodes.

Independent claim 13 recites a database system having a plurality of nodes, and instructions for enabling the database system to store first tuples in a first table distributed across the plurality of nodes, store second tuples in a second table distributed across the plurality of nodes, partition the first and second tuples into plural portions, redistribute the first and second tuples to the plurality of nodes according to the partitioning, and hash join the first and second tuples to produce result tuples as the first and second tuples are being redistributed to the plurality of nodes.

Independent claim 23 recites an article comprising a medium storing instructions for enabling a processor-based system to store first tuples in a first table in a database system, store second tuples in a second table in the database system, partition the first and second tuples into plural portions, redistribute the first and second tuples to plural nodes of the database system

according to the partitioning, and hash join the first and second tuples to produce result tuples as the first and second tuples are being redistributed to the plural nodes.

A hash join is a join operation that is performed using a hash table. For a value, **x**, a hash function, **f**, maps **x** to another value, **y**. The value **x** may be a number, a string, a spatial object, and so on. The value **y** is called the hash value of **x**. Specification, p. 8, lines 13-16.

A hash table is a table with many entries. Each entry deals with a specific hash value. Those values whose hash values are the same each end up in the same entry of the hash table. The hash table is thus a data structure for organizing data. Hash joins utilize hash functions and hash tables. Specification, p. 8, lines 19-22.

Traditionally, a parallel hash join is performed in two phases. First, tuples of one relation (known as a build relation) are redistributed to the nodes where the join will run. The tuples are added to the in-memory hash tables as they arrive at the nodes. Next, the tuples of a second relation (the probe relation) are redistributed to the same nodes. The hash tables built in the first phase are probed for each tuple of the probe relation that arrives at the nodes. However, this two-phase hash join technique is usually inefficient. Specification, p. 8, line 23-p. 9, line 8.

In contrast, according to one embodiment, a parallel hash ripple join algorithm performs join operations as the tuples are being redistributed to the nodes 10. Accordingly, join results may be available much sooner than with traditional join algorithms. Specification, p. 9, lines 15-17.

Tuples, or rows, of tables A and B are initially stored at the nodes according to some partitioning strategy, such as hash partitioning, range partitioning, or round-robin partitioning. The partitioning strategy typically attempts to distribute the tuples for a given table evenly across all available nodes of the relational database. Specification, p. 9, lines 18-22.

According to one embodiment, the parallel hash ripple join algorithm re-partitions the tuples 12 (Fig. 2). The tuples 12 are partitioned such that the tuples 12 for which the attributes 13 being compared during the join operation have identical values end up on the same node 10. The tuples 12 for each table 14 are thus redistributed to the nodes 10 to "localize" the join processing. Specification, p. 9, line 23-p. 10, line 2.

During redistribution of the tuples, each node receives tuples 12 from each of tables A and B, one after another. Since the tuples 12 are used for the join operation, the join operation may be performed as the tuples 12 arrive at the node, which enables result tuples to become available earlier than with conventional two-phase techniques. Specification, p. 11, lines 20-23.

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

A. Claims 1-4, 13-24, and 33-35 Are Rejected Under 35 U.S.C. § 103 Over Urhan, "XJoin: Getting Fast Answers from Slow and Bursty Networks," CS-TR-3994 in View of Kashyap.

B. Claims 5-12 and 25-32 Are Rejected Under 35 U.S.C. § 103 over Urhan in View of Kashyap and DeWitt, "Parallel Sorting on a Shared-Nothing Architecture Using Probabilistic Splitting," PDIS.

VII. ARGUMENT

A. Claims 1-4, 13-24, and 33-35 Are Rejected Under 35 U.S.C. § 103 Over Urhan, "XJoin: Getting Fast Answers from Slow and Bursty Networks," CS-TR-3994 in View of Kashyap.

1. Claims 1, 2, 4, 13, 14, and 16-24.

Independent claim 1 recites a method that comprises the acts of partitioning first tuples in a first table and second tuples in a second table into plural portions, redistributing the first and second tuples to plural nodes according to the partitioning, and hash joining the first and second

tuples to produce result tuples *as the first and second tuples are being redistributed to the plural nodes*.

Claim 1 was rejected by the Examiner as being obvious over Urhan and Kashyap. 5/6/2004 Office Action at 4. As conceded by the Examiner, Urhan does not disclose redistributing the first and second tuples to plural nodes according to the partitioning. *Id.* at 5. In view of this concession, the Examiner made a factual error in stating that Urhan teaches hash joining first and second tuples to produce result tuples *as the first and second tuples are being redistributed to the plural nodes. See id.* at 4. If Urhan does not teach the redistribution of first and second tuples to plural nodes, then it would be impossible for Urhan to teach hash joining of the first and second tuples *as the first and second tuples are being redistributed to the plural nodes*. In view of the factually erroneous statement made by the Examiner that Urhan discloses the hash joining act of claim 1, the obviousness rejection of claim 1 over Urhan and Kashyap is defective.

Moreover, the Examiner has failed to establish that there existed any motivation or suggestion to combine the teachings of Urhan and Kashyap. In fact, a careful reading of the references will reveal that there existed no such motivation or suggestion to combine, as the proposed modification of Kashyap based on the teachings of Urhan being urged by the Examiner would render the system of Kashyap inoperative for its intended purpose. "If a proposed modification would render the prior art invention being modified unsatisfactory for its intended purpose, then there is no suggestion or motivation to make the proposed modification. *In re Gordon*, 733 F.2d 900, 221 USPQ 1125 (Fed. Cir. 1984)." MPEP § 2143 (8th ed., Rev. 2) at 2100-131.

For efficient parallel processing, Kashyap states that each bucket in the hash table of each instance of a hash-join operator should ideally receive approximately the same amount of data. Kashyap, 2:3-5. Kashyap notes, however, that such an ideal distribution may not occur because the distribution of data values in "build" and "probe" tables (the build and probe tables are joined by the hash-join operator) is skewed or unevenly distributed. The hash-join mechanism described in Kashyap is designed to address the skew (uneven distribution) of either the "build" table or "probe" table.

To this end, Kashyap describes a partitioning technique for distributing records retrieved from a table among a first level of instances of a hash-join operator 28 (Kashyap, 6:15-19). However, to address the skew (uneven distribution) problem, the hash-join operator 28 of Kashyap must wait until *all* records of a build table 20 have been distributed into buckets of hash tables before any hash-join operation is performed. *See* Kashyap, Figures 3, 4A-4B, columns 5-12. The distributions of records of the build table 20 and of the probe table 22 are performed in sequence--the build table 20 is first distributed into buckets of a hash table, and after *all* records of the build table 20 have been distributed, the probe table 22 is distributed. *See* Kashyap, 9:12-24. Such a sequential distribution of records of the build table 20 and the probe table 22 of Kashyap is required to enable skew detection. *See* Kashyap, 6:57-59. As explained by Kashyap, skew is detected if a bucket receives too many records of the build table 20. When skew is detected, records of the build table 20 are re-routed to a second level of instances of the hash-join operator 32 (rather than to the first level of instances of the hash-join operator 28). *See* Kashyap, 7:42-52. Modifying Kashyap to enable incremental production of results during distribution of the records of the build and probe tables, as proposed by the Examiner, would render the Kashyap hash-join mechanism inoperative for its intended purpose, namely to enable skew or

uneven distribution detection. Therefore, no motivation or suggestion existed to combine Kashyap and Urhan in the manner proposed by the Examiner.

Moreover, one objective of the XJoin approach as described in Urhan is based on the principle of producing results incrementally as they become available so that early delivery of initial answers can be achieved. A further objective of the XJoin approach of Urhan is that the approach allows progress to be made even when one or more sources experience delays. As discussed by Urhan, when a tuple arrives at one of its inputs, the tuple is inserted into a hash table for that input and then immediately used to probe the hash table of the other input. Urhan, p. 4. "Thus, even if one source becomes temporarily blocked, the operator can still produce results." *Id.* This teaching of Urhan is inconsistent with the technique employed in Kashyap, where all records of a first table (the build table 20) must first be distributed to buckets among first-level instances and second-level instances of a hash-join operator 28 *prior* to the distribution of records of a second table (probe table 22). As discussed above, Kashyap performs this sequential distribution of records from two tables for handling skew in the distribution of records. If skew occurs, such as when a bucket is receiving too great a share of build records, Kashyap teaches that subsequent records are re-routed to a second level of instances of a hash-join operator (to a "spilled" bucket). Kashyap, 7:5-56. Following the distribution of all build records (from the first table or build table 20), distribution of the probe records (from the second table or probe table 22) is mapped according to whether spilling has occurred. Kashyap, 9:32-49. In other words, distribution of records of the second table are dependent upon distribution of records of the first table (based on whether spilling has occurred or not). Therefore, the objective sought by the XJoin technique of Urhan would be defeated by the technique employed in Kashyap, since early production of answers would not be achieved if the Urhan join system has

to wait for all records of one table to be distributed prior to distribution of records from a second table. Thus, combining the teachings of Kashyap into Urhan would also defeat the intended purpose of Urhan. Therefore, no motivation or suggestion existed to combine Urhan and Kashyap for this further reason. A *prima facie* case of obviousness has thus not been established with respect to claim 1 for the foregoing reasons.

In addition, the obviousness rejection is defective for the additional reason that even if Urhan and Kashyap can be properly combined, the hypothetical combination of references fails to teach or suggest *all* elements of claim 1. *See* M.P.E.P. § 2143 at 2100-129. As conceded by the Examiner, Urhan fails to disclose redistributing first and second tuples to plural nodes according to a partitioning. As noted above, Urhan also fails to disclose hash joining the first and second tuples to produce result tuples *as the first and second tuples are being redistributed to the plural nodes.*

Although the Examiner cited Kashyap as teaching the redistributing of first and second tuples to plural nodes according to the partitioning, the Examiner failed to explain how either Urhan or Kashyap teaches or suggests hash joining first and second tuples to produce result tuples *as the first and second tuples are being redistributed to plural nodes.* Urhan cannot perform such a hash joining task because Urhan does not disclose the redistribution of tuples to plural nodes. On the other hand, because Kashyap teaches that all records of the build table 20 (a first table) are distributed *prior* to distribution of the probe table 22 (the second table), *see* Kashyap 9:12-23, Kashyap cannot possibly teach or suggest hash joining first and second tuples to produce result tables *as the first and second tuples are being redistributed to the plural nodes.*

Therefore, even if Urhan and Kashyap can be properly combined, the hypothetical combination of the references fails to teach or suggest each and every element of claim 1. A *prima facie* case of obviousness has not been established for this additional reason.

Independent claim 13 is also not disclosed or suggested by the asserted combination of Urhan and Kashyap. Claim 13 recites a database system that includes a plurality of nodes and instructions for enabling the database system to store first tuples in a first table distributed across the plurality of nodes, store second tuples in a second table distributed across the plurality of nodes, partition the first and second tuples into plural portions, redistribute the first and second tuples to the plurality of nodes according to the partitioning, and hash join the first and second tuples to produce result tuples as the first and second tuples are being redistributed to the plurality of nodes. As explained above, the asserted combination of Urhan and Kashyap does not disclose or suggest the hash joining of first and second tuples of first and second tables to produce result tuples *as the first and second tuples are being redistributed to the plurality of nodes*. The obviousness rejection against claim 13 is defective for at least the following reasons: (1) there existed no motivation or suggestion to combine the Urhan and Kashyap to achieve the claimed subject matter; and (2) the hypothetical combination of Urhan and Kashyap fails to disclose or suggest *all* elements of claim 13.

Independent claim 23 is similarly allowable over the asserted combination of Urhan and Kashyap.

For the foregoing reasons, it is respectfully requested that the final rejection of the above claims be reversed.

2. Claims 3 and 15.

Claims 3 and 15 depend from independent claims 1 and 13, respectively, and thus are allowable for at least the same reasons.

Moreover, claim 3 recites retrieving result tuples at random. The Examiner cited page 4, Figs. 1-2, page 6, Figs. 3-4, and pages 2-3 as teaching this feature of claim 3. 5/6/2004 Office Action at 5. There is no indication or suggestion anywhere in the cited passages of Urhan of retrieving result tuples (produced from hash joining first and second tuples) *at random*.

A *prima facie* case of obviousness of claim 3 and 15 fails for this further reason. Therefore, reversal of the final rejection of the above claims is respectfully requested.

B. Claims 5-12 and 25-32 Are Rejected Under 35 U.S.C. § 103 over Urhan in View of Kashyap and DeWitt, "Parallel Sorting on a Shared-Nothing Architecture Using Probabilistic Splitting," PDIS.

1. Claims 5-12 and 25-32.

Dependent claims 5-12 and 25-32 were rejected as being obvious over the asserted combination of three references: Urhan, Kashyap, and DeWitt. Because no motivation or suggestion existed to combine the teachings of Urhan and Kashyap, the three-way combination of Urhan, Kashyap, and DeWitt also fails. Furthermore, because the hypothetical combination of Urhan and Kashyap fails to teach or suggest all elements of base claims 1, 13, and 23, the hypothetical combination of Urhan, Kashyap, and DeWitt would also fail to teach or suggest all elements of the dependent claims.

For the foregoing reasons, reversal of the final rejection of the above claims is respectfully requested.

VIII. CONCLUSION

In view of the foregoing, reversal of all final rejections and allowance of all pending claims is respectfully requested.

Respectfully submitted,

Date: 10-27-04

Dan C. Hu
Registration No. 40,025
TROP, PRUNER & HU, P.C.
8554 Katy Freeway, Suite 100
Houston, TX 77024
Telephone: (713) 468-8880
Facsimile: (713) 468-8883

APPENDIX OF CLAIMS

The claims on appeal are:

1. A method comprising:

storing first tuples in a first table in a database system;

storing second tuples in a second table in the database system;

partitioning the first and second tuples into plural portions;

redistributing the first and second tuples to plural nodes according to the partitioning; and

hash joining the first and second tuples to produce result tuples as the first and second tuples are being redistributed to the plural nodes.

2. The method of claim 1, further comprising:

retrieving the result tuples once the hash join is performed.

3. The method of claim 1, further comprising:

retrieving the result tuples at random.

4. The method of claim 1, hash joining the first and second tuples to produce result tuples as the first and second tuples are being redistributed to the plural nodes further comprising:

producing result tuples at one of the plural nodes; and

simultaneously producing result tuples at a second of the plural nodes.

5. The method of claim 1, wherein redistributing the first and second tuples to plural nodes comprises redistributing based on split vectors containing predefined ranges.

6. The method of claim 5, wherein partitioning the first and second tuples into plural portions comprises:

partitioning first and second tuples into hash tables in each node.

7. The method of claim 6, wherein hash joining the first and second tuples comprises:

allocating a portion of a memory to a first hash table;

allocating a second portion of the memory to a second hash table; and

hash joining first tuples in the first hash table with second tuples in the second hash table.

8. The method of claim 7, wherein hash joining the first and second tuples comprises:

determining that the portion of the memory allocated to the first hash table is full;

allocating a stable storage to the first hash table; and

storing first tuples in the stable storage.

9. The method of claim 8, further comprising:

continuing to store second tuples in the second hash table; and

hash joining second tuples in the second hash table with first tuples in the first hash table.

10. The method of claim 9, further comprising:

determining that the second portion of the memory allocated to the second hash table is full;

allocating a second stable storage to the second hash table;

storing second tuples in the second stable storage; and

hash joining second tuples in the second stable storage with first tuples in the first hash table.

11. The method of claim 10, wherein hash joining the first and second tuples comprises:

generating a third hash table once all first tuples and second tuples are redistributed to each node;

retrieving one of the first tuples from the stable storage;

hash joining the one of the first tuples with tuples in the second hash table; and

storing the one of the first tuples in the third hash table.

12. The method of claim 11, further comprising:

retrieving one of the second tuples from the second stable storage; and

hash joining the one of the second tuples with tuples in the third hash table.

13. A database system comprising:

a plurality of nodes; and

instructions for enabling the database system to:

store first tuples in a first table distributed across the plurality of nodes;

store second tuples in a second table distributed across the plurality of nodes;

partition the first and second tuples into plural portions;

redistribute the first and second tuples to the plurality of nodes according to the partitioning; and

hash join the first and second tuples to produce result tuples as the first and second tuples are being redistributed to the plurality of nodes.

14. The database system of claim 13, wherein the result tuples are available once the hash join is performed.

15. The database system of claim 13, wherein the result tuples are available at random.

16. The database system of claim 13, wherein each node comprises a memory, and wherein the instructions further partition the first and second tuples into plural portions by:

partitioning first tuples into first hash tables; and

partitioning second tuples into second hash tables, wherein the hash tables are in the memory.

17. The database system of claim 16, wherein the instructions further:

allocate a portion of the memory to the first hash table;

allocate a second portion of the memory to the second hash table; and

hash join first tuples in the first hash table with second tuples in the second hash table.

18. The database system of claim 17, wherein the instructions further:

determine that the portion of the memory allocated to the first hash table is full; and

store first tuples in a stable storage.

19. The database system of claim 18, wherein the instructions further:

continue to store second tuples in the second hash table; and

hash join second tuples in the second hash table with first tuples in the first hash table.

20. The database system of claim 19, wherein the instructions further:

determine that the second portion of the memory allocated to the second hash table is full;

allocate a second stable storage to the second hash table;

store second tuples in the second stable storage; and

hash join second tuples in the second stable storage with first tuples in the first hash table.

21. The database system of claim 20, wherein the instructions further:

generate a third hash table once all first tuples and second tuples are redistributed to each node;

retrieve one of the first tuples from the stable storage;

hash join the one of the first tuples with tuples in the second hash table; and

store the one of the first tuples in the third hash table.

22. The database system of claim 21, wherein the instructions further:

retrieve one of the second tuples from the second stable storage; and

hash join the one of the second tuples with tuples in the third hash table.

23. An article comprising a medium storing instructions for enabling a processor-based system to:

store first tuples in a first table in a database system;

store second tuples in a second table in the database system;

partition the first and second tuples into plural portions;

redistribute the first and second tuples to plural nodes of the database system according to the partitioning; and

hash join the first and second tuples to produce result tuples as the first and second tuples are being redistributed to the plural nodes.

24. The article of claim 23, further storing instructions for enabling a processor-based system to:

retrieving the result tuples once the hash join is performed.

25. The article of claim 24, further storing instructions for enabling a processor-based system to:

redistribute based on split vectors containing predefined ranges.

26. The article of claim 25, further storing instructions for enabling a processor-based system to:

partition first and second tuples into hash tables in each node.

27. The article of claim 26, further storing instructions for enabling a processor-based system to:

allocate a portion of a memory to a first hash table;

allocate a second portion of the memory to a second hash table; and

hash join first tuples in the first hash table with second tuples in the second hash table.

28. The article of claim 27, further storing instructions for enabling a processor-based system to:

determine that the portion of the memory allocated to the first hash table is full; and

store first tuples in a stable storage.

29. The article of claim 28, further storing instructions for enabling a processor-based system to:

continue to store second tuples in the second hash table; and

hash join second tuples in the second hash table with first tuples in the first hash table.

30. The article of claim 29, further storing instructions for enabling a processor-based system to:

determine that the second portion of the memory allocated to the second hash table is full;

allocate a second stable storage to the second hash table;

store second tuples in the second stable storage; and

hash join second tuples in the second stable storage with first tuples in the first hash table.

31. The article of claim 30, further storing instructions for enabling a processor-based system to:

generate a third hash table once all first tuples and second tuples are redistributed to each node;

retrieve one of the first tuples from the stable storage;

hash join the one of the first tuples with tuples in the second hash table; and

store the one of the first tuples in the third hash table.

32. The article of claim 31, further storing instructions for enabling a processor-based system to:

retrieve one of the second tuples from the second stable storage; and

hash join the one of the second tuples with tuples in the third hash table.

33. The method of claim 1, wherein storing the first tuples in the first table comprises distributing the first tuples across the plural nodes of the database system, and wherein storing the second tuples in the second table comprises distributing the tuples across the plural nodes.

34. The method of claim 33, wherein redistributing the first and second tuples comprises redistributing the first and second tuples to the plural nodes of the database system.

35. The article of claim 23, wherein storing the first tuples in the first table comprises storing the first tuples in the first table distributed across the plural nodes of the database system, and wherein storing the second tuples in the second table comprises storing the second tuples distributed across the plural nodes of the database system.